National Research Council Canada
Conseil national de recherches Canada

ERB-1037

Institute for Information Technology
Institut de technologie de l'information

NRC·CNRC

Discrete Event Simulation using Object-oriented Programming: Application to Manufacturing

H. Atabakhsh and A.W. Chan
May 1992

Canada

NRC No. 33188

The Institute for Information Technology (IIT) is committed to assisting industry across Canada to achieve excellence in information technology software and systems by promoting, supporting, and undertaking focused, industry-oriented research and development programs in these areas. To achieve this mission, IIT works in collaboration with industrial, educational, government, and other organizations.

There are four elements to the Institute's research program: software engineering and knowledge-based systems as part of software technology; sensor-based automation and systems integration as part of systems technology.

There are four laboratories in the Institute, with each laboratory responsible for one research element.

The Institute invites inquiries with regard to its mission and research program and will be pleased to discuss any proposal for research or collaborative activity. Contact the Industrial Liaison Office at (613) 993-2491.

L'Institut de technologie de l'information (ITI) est chargé d'aider l'industrie partout au Canada à atteindre l'excellence en matière de logiciels et de systèmes de la technologie de l'information en faisant la promotion de programmes de recherche-développement focalisés et tournés vers l'industrie, de même qu'en appuyant et en entreprenant de tels programmes. Pour s'acquitter de sa mission, l'ITI collabore avec des organisations de l'industrie, de l'enseignement et du gouvernement et avec d'autres organismes.

Le programme de recherche de l'ITI comporte quatre éléments : génie logiciel et systèmes à base de connaissances dans le cadre de la technologie des logiciels; automatisation à base de capteurs et intégration des systèmes dans le cadre de la technologie des systèmes.

L'Institut compte quatre laboratoires; chaque laboratoire est chargé d'un domaine de recherche.

L'Institut se fera un plaisir de répondre à toute question concernant sa mission et son programme de recherche, et de discuter de tout projet de recherche ou d'activité conjointe. Communiquer avec le Bureau de liaison industrielle au (613) 993-2491.

Additional copies are available free of charge from:

Editorial Office, Room 301
Institute for Information Technology
A.G.L. McNaughton Building (M-50)
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R6

Des exemplaires supplémentaires peuvent être obtenus gratuitement à l'adresse suivante :

Bureau des publications, Pièce 301
Institut de technologie de l'information
Immeuble A.G.L. McNaughton (M-50)
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R6

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

Discrete Event Simulation using Object-oriented Programming: Application to Manufacturing

H. Atabakhsh and A.W. Chan
May 1992

Copyright 1992 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Contents

Abstract/Résumé v

1. Introduction 1

2. Object-oriented Implementation of Discrete Event Simulation 1
   - 2.1. Object-oriented Programming Concepts 1
   - 2.2. Implementation of Discrete Event Simulation 2
   - 2.3. Implications and Advantages of using Object-oriented Programming 3

3. An Example in Manufacturing 4
   - 3.1. Description of the Example 4
   - 3.2. Basic Entities in the Example — Smalltalk Objects 4
   - 3.3. Basic Events Scheduled — Smalltalk Methods 6
   - 3.4. Execution 7
   - 3.5. Graphical User Interface 7

4. Comparison with the Multiprocessing Approach in Smalltalk-80 9

5. Conclusions and Future Directions 10

Acknowledgment 10

References 10

Appendix A: List of Entities and their Attributes for the Model 11

Appendix B: Hierarchy of Smalltalk Objects for the Enterprise Model 13

Appendix C: Flowcharts for the Events 15
   - C1. Event Hierarchy 15

Appendix D: Description of the Flowcharts 16
   - D1. Event Sequence — Part 1 16
   - D2. Event Sequence — Part 2 17
   - D3. Event — Review Schedule 18
   - D4. Event: Issue Task to Material Handler 20
   - D5. Event: Start of Material Handling Task 22
   - D6. Event: End of Material Handling Task 24
   - D7. Event: Release Source Device 25
   - D8. Event: Sink Device Receives Part 27
   - D9. Event: Start of Machining Operation 28
   - D10. Event: End of Machining Operation 29
   - D11. Event: End of Non-machining Operation 31
   - D12. Documentation and Smalltalk Programs Cross-reference List 32

Appendix E. Input Data for Initializing the Enterprise Model Example 33
   - E1. Schedules for Machines 33
   - E2. Bill of Operations for Jobs 33
   - E3. Other Objects 34

Figures

1. User interface; initial state of the example. 8

2. User interface; material handlers moving parts between machines and storage areas. 8

Abstract

Discrete-event simulation following the traditional approach of event scheduling has been implemented using the object-oriented paradigm. Concepts and terminology in object-oriented programming are briefly reviewed and implications of using this paradigm for discrete event simulation are discussed. To better illustrate the concepts, an example in the area of discrete parts manufacturing has been implemented in Smalltalk-80. The approach used is, however, generic and can be implemented in other object-oriented programming environments and applied to other domains in discrete event simulation.

Résumé

La simulation à événements discrets, suivant l'approche traditionnelle d'établissement du calendrier d'événements, a été implantée par le biais de la programmation orientée objet. Les auteurs passent brièvement en revue les notions et la terminologie de la programmation orientée objet, puis décrivent les conséquences de ce type de programmation pour la simulation à événements discrets. Pour permettre de mieux saisir ces notions, un exemple a été implanté en Smalltalk 80 dans le domaine de la fabrication de pièces détachées. Toutefois, l'approche utilisée est générique et peut être implantée dans d'autres environnements de programmation orientée objet, et être appliquée à d'autres secteurs de la simulation à événements discrets.

1. Introduction
2. Object-oriented Implementation of Discrete Event Simulation

1. Introduction

For over a decade the Systems Integration Laboratory has been collaborating with various industrial sectors (mining, metal production, transportation, etc.) on modeling and simulation of industrial plants. Simulation models were used to investigate proposed changes to plant operations or equipment, the impact of proposed plant expansions or the design of new plants or transportation systems. An example of a simulation package that evolved from these activities is ANEVENT [1], which is based on the simulation language GASP II and FORTRAN. ANEVENT provides a structure for building interactive, discrete event computer models.

In the last few years, the object-oriented paradigm has been recommended as a development tool for various programming applications. The main reason for this choice is the reusability and flexibility that is achieved through object-oriented programming. Programming is a repetitive task; reusability is therefore very desirable. Object-oriented programming provides techniques for capturing commonalities within groups of related data structure implementations and techniques for isolating users from internal module representations.

In object-oriented environments, the system is constructed around objects (data) which are grouped into classes, depending on their commonalities. Properties and methods (functions) are inherited by objects, in a hierarchy of classes. Objects communicate through message passing or function calls and are the key to system modularity because they promote reusability and extendibility of code.

The top-down functional refinement approach is less flexible, reusable, and extendible than the object-oriented approach for various reasons. In the former, the data structure aspect is sometimes neglected, which implies little reusability because objects are shared by various functions. The emphasis is on functions, which are less stable than data structures, i.e., if the system changes, the software must change. If software is developed around data structures, however, they can be used again if the system changes. Also, there is less flexibility in the top-down approach because the actions are ordered from the beginning.

Simulation is one of the application domains of object-oriented programming. Programs are written around the objects whose behavior is being simulated. The software is constructed as structured collections of abstract data types, i.e., the data structures are described by a list of services available to them and their properties, not by their implementation. For a description of discrete event simulation, the reader may refer to ref. 2.

The above considerations led to the development of a discrete event simulation package implemented in an object-oriented programming language, as described in this report. Some of the reasons why Smalltalk-80 was chosen for the implementation are: Smalltalk has most of the features an object-oriented package should have (encapsulation, dynamic binding, polymorphism, garbage collection, etc.); it is a relatively mature language; it has a large library of classes; it is a pure language, which might be a better choice for starting to program in an object-oriented environment. There has been an attempt to keep the Smalltalk program very generic, however, so that it can be translated into other object-oriented languages such as C++, which is also widely used.

In this report, after a brief review of important concepts and terminology in object-oriented programming, generic aspects of discrete event simulation are described together with their implementation in the object-oriented paradigm. Implications and advantages of using the object-oriented paradigm for discrete event simulation are discussed. An example in discrete parts manufacturing is presented to illustrate the implementation of the traditional approach to discrete event simulation in Smalltalk-80. Finally, a comparison is made with the multiprocessing approach in Smalltalk-80 proposed by Goldberg and Robson [3], and future research directions are proposed.

2. Object-oriented Implementation of Discrete Event Simulation

2.1. Object-oriented Programming Concepts

In this section some of the important concepts and terminology in object-oriented programming are briefly reviewed. For a more detailed discussion on object-oriented programming, readers may refer to refs. 3–5. Object-oriented programming systems view the world as a collection of discrete objects which act in a common environment by exchanging messages.

- Objects and classes: An object is a group of data along with procedures (usually called methods) to access and modify the data. Each object is an instance of its respective class. A class is a means of defining an object in terms of its abstract type. A software object can represent either a physical or a conceptual object.

- Encapsulation: A key issue is that each object has only private data, which can be manipulated only via the procedures that are provided as part of the object. This property of binding both data and code into a single object (or class) is referred to as encapsulation. An object can therefore be used without regard to how it was actually implemented internally. The data are stored in instance variables which can be pointers to other objects.

- Data Abstraction: Classes of objects are described by their external features rather than by their physical representations. The features are the methods applicable to objects of the class and abstract properties of these methods. These operations (methods) appear as the way object classes are characterized. The essential difference with classical techniques based on procedural decomposition is that functions are attached to data structures rather than the reverse. Objects are therefore described as implementations of abstract data types.

- Inheritance: Classes are organized into hierarchies in which a class that is subordinate to another automatically inherits all the instance variables and methods of the superior class. The objects that are instances of a subclass recognize all the messages and instance variables of their superclass as well as their own. This concept allows the addition of complexity and functionality to simple objects as necessary by creating and maintaining subclasses of existing object classes. It also minimizes data specification and redundancy.

- Message passing: In most object-oriented languages, messages are the only means for the objects to communicate with each other (in some object-oriented languages, e.g., Eiffel and CLOS, they communicate through function calls). A message to an object requests the execution of a procedure. A message invokes a method (procedure) in the receiving object to produce the required behavior.

- Dynamic binding and polymorphism: These properties enable the definition of flexible program entities that may refer to instances of various classes at run time, all of which offer an operation with the same external specification but different implementations. The application of an operation to the entity will result in the appropriate implementation being selected, depending on the particular object associated with the entity at the time the operation is executed. In other words, different objects can accept the same message but respond differently. In the next section how discrete event simulation can be implemented in an object-oriented environment will be discussed.

2.2. Implementation of Discrete Event Simulation

A discrete event model represents behavior in terms of dependent variables that change discretely at specified instances of simulated time. A discrete change in the status of the system being modeled is called an event, and the time at which an event occurs is called the event time. An event calendar usually stores the list of events with their associated event times. Activities are initiated and terminated with events. The event is the point in time at which state changes in the modeled system take place. The dynamic behavior of the modeled system is obtained by allowing the events to occur in their natural sequence, thereby causing state changes to occur. Typically, the occurrence of one event initiates the occurrence of one or more other events. Time is advanced from event to event until some stopping condition is reached. The model replicates the activities that the entities engage in, where each activity begins and ends with an event and has a finite time duration. Each entity may compete with other entities in the model for utilization of scarce resources.

To implement discrete event simulation in an object-oriented paradigm, an object that functions as an event calendar is needed. This object is called here the time stack. This is a stack of future events ordered by their event times. When the simulation clock is advanced to the next event time, the corresponding object is informed, and the corresponding event is triggered. Each event in the time stack is itself an object and contains the data associated with a particular event which is yet to happen. When a new event is created, it is inserted into the time stack at the correct location depending on the time the event is due, such that the next event to happen is always at the head of

the queue. The simulation proceeds by removing events, in sequence, from the top of the time stack, processing them, and then updating the simulation clock to the time of the next event. This processing usually results in new events being created and added to the stack, thereby causing further processing to occur and the simulation time to advance.

The basic entities needed for implementing discrete event simulation are the event queue (time stack), the events, and event simulation time. Each of these entities is implemented as an "object" in object-oriented terms.

The time stack is implemented as an instance of class "Sorted Collection" and contains events sorted according to the simulation time they occur. An element from the stack is an instance of class "Association," where the key is the simulation time at which the event should occur, and the value is an instance of class "Array" containing the event (implemented as a message, i.e., method), and the receiver of the message (which is an object).

These objects are generic and can be used for any application in discrete event simulation. For a given application, other objects will be needed specific to that application. For example, as described in Section 3, in the manufacturing area objects such as machines, material handlers, storage areas, parts, and products need to be implemented together with the objects such as the time stack, events, and the event simulation time.

In the following section, some of the implications and advantages of using object-oriented programming for discrete event simulation are discussed.

2.3. Implications and Advantages of using Object-oriented Programming

Object-oriented programming is increasingly used for discrete event simulation [6–14]. Use of the object-oriented paradigm in simulation has the potential for many advantages not available through other methods. Some of them are given in this section.

Representing entities as individual objects allows a natural one-to-one correspondence between real-life objects and software objects. It makes the software conceptually simpler to understand. As an example, in a manufacturing application, real-world entities (such as machines, material handlers, storage areas, parts, and products), as well as simulation entities (the event queue, event time, simulation clock, etc.), can each be represented by an object or class of objects.

The concept of classes permits the representation of interactions in multiple levels of abstraction. This should allow, for example, the simulation of an entire enterprise or a manufacturing cell in the enterprise, with relative ease.

Simulation of complete systems leads to large-scale software development. The object-oriented approach has the potential to reduce life-cycle costs and allows better maintainability and extendibility, through reusability of software objects [5].

Through polymorphism, different objects can accept the same message but respond differently. This facilitates the association of objects that experience a similar event through the same message. As an example, a message corresponding to an event is sent to an object "Device," and, depending on the type of device (machine or storage area), a different set of actions is taken. Another example would be sending a message to an object belonging to the class "Operation" and, depending on the type of operation (setup, machining, or teardown), a different set of actions is triggered. This prevents writing any redundant code, because the same code can be used differently by various objects.

Following the object-oriented paradigm has the advantage of preserving the bulk of developed code for general use in model building. Each model building problem then performs the particular functions that are of interest at that time. The object definitions remain independent of the functions of the system being modeled. The objects corresponding to discrete event simulations, for example, are generic and can be used in various applications. This is an example of reusability of code. For each application, objects specific to the application domain are implemented and used alongside the discrete event simulation objects, instead of starting from scratch for each application.

As the capabilities of the system are enhanced, the inheritance capabilities of object-oriented programming can be used to create subclasses of the generic objects of the system. For example, in a manufacturing application, the modeling of material handlers will be extended to include lift trucks, fork lifts, AGVs, etc. These enhancements are readily achievable in object-oriented languages because the more specific

subclasses automatically inherit the capabilities of their more general parents.

Most object-oriented packages include a good graphical user interface and provide mechanisms for the development of visual interactive simulation. The output of simulation can therefore be visualized on the screen as the model is running. This is especially interesting if animation is used. The movement of parts or resources on the shop floor can, for example, be simulated and visualized.

The debugging process in object-oriented languages is somewhat different from procedural languages. A single object can be debugged and tested, without affecting any other parts of the code.

In the next section, an example in the manufacturing application domain is presented in order to better illustrate the concepts discussed above.

3. An Example in Manufacturing

3.1. Description of the Example

The discrete event simulation written in Smalltalk-80 and described in the previous sections has been tested in the Manufacturing Enterprise Modeling Project within the Software for Integrated Manufacturing Consortium (SIMCON). This project concerns developing a software system for assisting decision making in a manufacturing environment by providing a set of tools for the creation of models of manufacturing systems. These models will be used to improve the ways in which manufacturing facilities are managed and operated by showing the user how changes in one or more of the parameters (types and quantities of resources, information flows, etc.) would affect the performance of the manufacturing enterprise. For more details on this project, refer to ref. 15.

The simulation model has been tested using a number of examples in the Manufacturing Enterprise area. One of the typical examples is described here and the data used for the example are given in Appendix E.

It is an example of discrete parts manufacturing. Parts arrive in the system and are processed by a sequence of machines. Each machine has a corresponding schedule. Each entry in the schedule has information such as a scheduled start time, a scheduled end time, an operation identity and type, and a job identity. A machine might need certain fixtures before processing a part. Fixtures and parts are transported through the system by material handlers. Sometimes a given material handler might be busy and unable to perform its job immediately. When the scheduled time for a machine arrives, the corresponding operation is started. The information about a specific operation (predecessor, successor, device identity, machining time, material handler identity, etc.), can be found in the bill of operations corresponding to the job.

There are three types of operations:

- setup: the appropriate material handler, when idle, takes a fixture from the appropriate storage and sets up the appropriate machine. All the information needed is found in the data corresponding to the setup.
- machining: the given material handler takes a part from the appropriate storage to the appropriate machine, which must already have been set up for machining. All the necessary information is given in the Bill Of Operations.
- teardown: the given material handler tears down the given machine by taking the fixture to the appropriate storage area. The necessary information is found in the data corresponding to the teardown.

In the example considered here, there are three machines to schedule, plus two other machines that are used but not scheduled. There are three jobs, therefore three bills of operations, one for each job. There are also eight material handlers, two storage areas containing fixtures, and seven storage areas containing raw parts and final products. Resources such as material handlers are shared between machines (jobs). Sharing material handlers means that at certain times a machine must wait for a material handler to be free before a part can be processed on it or removed from it. In the case where the material handler is busy, the task is put in its task queue. Since the scheduled start times for the machines may overlap, the actual execution of operations on machines could take place in parallel.

3.2. Basic Entities in the Example — Smalltalk Objects

To implement discrete event simulation for this example, two set of entities need to be represented. The first set includes all entities related to the simulation

itself. These are the event queue (time stack), event time, and simulation clock. The second set includes all entities related to the application domain (i.e., to the specific example). By separating these two sets, the simulation objects can be used for other application domains and other examples without any modifications. The objects corresponding to entities in a given application domain will simply be added to the system. The nature of object-oriented programming easily allows this because the objects are independent entities communicating with one another only through message passing.

3.2.1. Entities Corresponding to the Simulation

The basic entities to be implemented for discrete event simulation are the event queue, the events, and the simulated event time. The event queue is implemented in Smalltalk-80 as an object called TimeStack and is an instance of class "SortedCollection." TimeStack contains events sorted according to the simulation time at which they occur. An element in the TimeStack is an instance of class "Association" where the key is the simulation time the event occurs and the value is an instance of class "Array" containing the event and the receiver of the event. Events are implemented as messages (Smalltalk methods), and their receivers are Smalltalk objects (receivers of the message). Events are therefore implemented as Smalltalk methods and depend on the application area. The events considered for the example in the manufacturing area are described in Section 3.3.

The SortedCollection is a convenient way of implementing this structure because it is automatically sorted by the key (in this case the event time), in increasing order. A description of the way the time stack is used to simulate the example is given in Section 3.4.

3.2.2. Entities Corresponding to the Application Domain

The enterprise is described in terms of a number of entities, whose interaction influences the behavior of the enterprise. Each entity is described by a set of attributes. Event attributes identify the type of event, event time, and other parameters required in executing the event.

The list of entities and their attributes is given in Appendix A.

Each entity in the enterprise model is implemented by a Smalltalk class, and its attributes are represented by "instance variables." The class hierarchy of Smalltalk objects and their instance variables, as implemented in our model, are given in Appendix B. Some of the important entities (objects) and their attributes are briefly described below. A more detailed list can be found in Appendix A.

- The entity Machine represents the resources used in the model and has attributes such as status (idle, setup in progress, waiting for part, machining, etc.), time the status was assumed, whether part is on the machine, and current job.

- The entity Storage Area represents the physical device that stores elements such as fixtures, raw parts, and final products. Attributes of this entity include status (full or not full), total number of elements it contains, and for each job, the number of parts, its status, etc.

- The Material Handler represents the physical entity that moves fixtures, raw parts, and final products between the devices (storage areas and machines). The information necessary for this type of entity is stored in attributes such as status (idle or busy), time status was assumed, which device the material handler is moving from, which device it is going to, its speed, and different durations such as time it takes for proceeding to source device and time for picking up a part.

- Operation represents a machining operation and has attributes such as preceding and succeeding operations, device identity, duration of identity, and status (e.g., not started, waiting for part, all parts completed, etc.).

- Setup corresponds to a setup operation and is described by attributes such as machine identity, duration of setup, and storage identity. Teardown corresponds to a teardown operation and has the same type of attributes as Setup.

- The entity Task represents a task which is created as soon as it must be issued to a material handler. The task is created by specifying its source device, sink device, and the material handler identity. A task also has attributes which store its creation time, start time, and end time.

- Schedule represents an entry in the schedule. Some of its attributes are scheduled start and end times, simulated start and end times, operation type, machine identity.

- Job, as its name indicates, represents a job in the enterprise model. It is described by its identity, total number of parts to produce, and its bill of operations identity.

- Bill of Operations is itself an entity in the model and has the attribute job identity. It is implemented as an instance of class OrderedCollection, with each element of the collection being an object Operation. All the other entities described above are implemented as instances of the class EnterpriseObject in order to group them all under one class. EnterpriseObject is itself an instance of Object (the super class of all classes in Smalltalk [3]) .

The entity machine-list representing the list of machines to be scheduled is implemented as an instance of Ordered Collection. The elements in the collection representing the list of machines, belong of course to the class Machine. There is also an object Cell, an instance of class Model (included in Smalltalk-80 [3]), which is used for the graphics user interface described in Section 3.5. Each of these entities is implemented as a Smalltalk class. Individual components of the system are instances of these classes. For example, Machine is a class and machine1, machine2, etc., are instances of the class Machine. Operation is also a class; operation1, operation2, etc., are instances of that class. The same type of implementation applies to other entities in the system (e.g., storage area, material handler, job, etc.). Each component is added to the model by creating an instance of its corresponding class and storing it in a Dictionary (an instance variable of a class called Cell) with the key being the symbol (or string) representing the component.

3.3. Basic Events Scheduled — Smalltalk Methods

The events are described in the flowcharts given in Appendix C and a description of each flowchart can be found in Appendix D. Only a brief description of each event is given here.

- The event "review schedule," reviews the schedule for the current machine to see if there are any more operations left to schedule. If yes, the start time for the operation is checked. If the time has not yet arrived, then this event is scheduled in the time stack to be executed when the corresponding time arrives. Otherwise, depending on the operation type, a task is issued to a certain material handler.

- "Issue Task to material handler," is itself an event. The status of the material handler is checked. If it is idle, then the event "start of material handling" is triggered to be executed. Otherwise, the task is inserted into the task queue of the material handler.

- In the event "start material handling," the status of the material handler is updated to busy, and the information about the task is also updated. Then, depending on the operation type (machining, setup, teardown), certain events are scheduled in the time stack.

- The event corresponding to the end of the material handling task updates information about the task and updates the material handler statistics. Then, if there are still tasks in the task queue of the material handler, the task is removed and the start of material handling task is triggered.

- The event "release source device" checks the device type which could be a machine or storage area. In the case of a storage area the bill of operations and storage statistics are updated. Otherwise the next task for this machine is checked to see if the previous machine is waiting to be unloaded, if the part in storage awaits processing, if all parts for this job are completed, or if no part for this job is ready. In each of these cases, certain information is updated and a certain task is triggered to be issued.

- The event "sink device receives part" updates storage area statistics if the device is a storage area. If the device is a machine, the bill of operations and machine attributes are updated, and the start of machining operation is triggered.

- The event "start of machining operation" updates the machine status, determines the duration of the machining operation, and schedules the end of machining operation event to occur after the duration of machining.

- "End of machining operation", after testing the next device type and testing whether the next machine is ready or the next storage is empty, triggers a certain

task to be issued to a corresponding material handler.

- The event "end of non-machining operation," when triggered, updates the scheduling information and the machine status and then triggers the event "review schedule."

Each of the events described above is implemented by one or more Smalltalk-80 methods.

3.4. Execution

The discrete event simulation reported here has been implemented in Smalltalk-80 (objectworks) on a Decstation 3100 under the Ultrix operating system. The approach used is based on traditional discrete event simulation. There have been other implementations of discrete event simulation in Smalltalk, such as the approach based on processes developed by Goldberg and Robson [3]. They indicate that their approach is only one of the many ways one can specify a simulation control framework for the Smalltalk-80 language. These two approaches are briefly compared in Section 4.

In the example described in this section, there is a list containing the set of machines to schedule. The execution starts by reviewing the schedules for these machines to see if the scheduled start time for one has arrived. If the scheduled time for one of the machines has arrived, then the event "ReviewSchedule" is triggered. Otherwise (i.e., if the scheduled start time has not arrived yet) the event ReviewSchedule for the given machine is scheduled in the event queue. The event ReviewSchedule is described by the flowchart 10 in Appendix C. As described by the flowcharts, this event triggers and schedules other events which in turn trigger and schedule other events in the event queue.

The event queue (called timeStack) is a Sorted Collection, whose elements have a key which is the time associated with an event and a value representing the event. The timeStack is sorted in increasing order of time, so that the first element (top of list) is the next event to trigger. The next event time is the time associated with this event.

The simulation time is updated to the next event time and then the events at that time are removed from the event queue and executed (triggered). This goes on until there are no more events left in the event queue.

A step-by-step execution is also possible. In this case, the execution stops after all events at the current simulated time (next event time) are executed. The execution must be resumed by the user as will be described in the next section. A trace of the execution can be output into a file. Visualization of the simulation is facilitated by a graphical user interface described in the next section.

algorithm for starting the execution:

if scheduled start time for machine has arrived then 1) trigger event ReviewSchedule (10) for the machine else:

2) schedule event ReviewSchedule (10) for the machine

3) if start time for neither machine has arrived and all are scheduled then: 4) check timestack and execute events at current simulation time

5) Return

else: 6) trigger event ReviewSchedule (10) for the machine

3.5. Graphical User Interface

There is a graphical user interface which helps visualize the execution of the Smalltalk program implementing the simulation of the Enterprise Model (see Fig. 1). It uses the model-view-controller (MVC) paradigm supported by Smalltalk [3] which facilitates building interactive applications by providing a mechanism for capturing information and operations in a particular problem domain (Model), graphic display of the domain information (View), and control of the user – problem domain interaction (Controller).

The physical entities in the system such as resources (machines), storage areas (containing fixtures, raw parts or final products), and material handlers are represented by graphic icons. The icons are labeled by the identity of the corresponding entity. The icons representing machines include a label indicating the status of the machine. The icons representing the storage areas have a label showing the number of elements (parts or products) they contain.

During the simulation, status changes and number of elements in the storage areas are reflected interactively, so that the user can follow the progression of the simulation. The icons representing the material handlers move from one device to another, animating the movement of parts and fixtures (see Fig. 2).

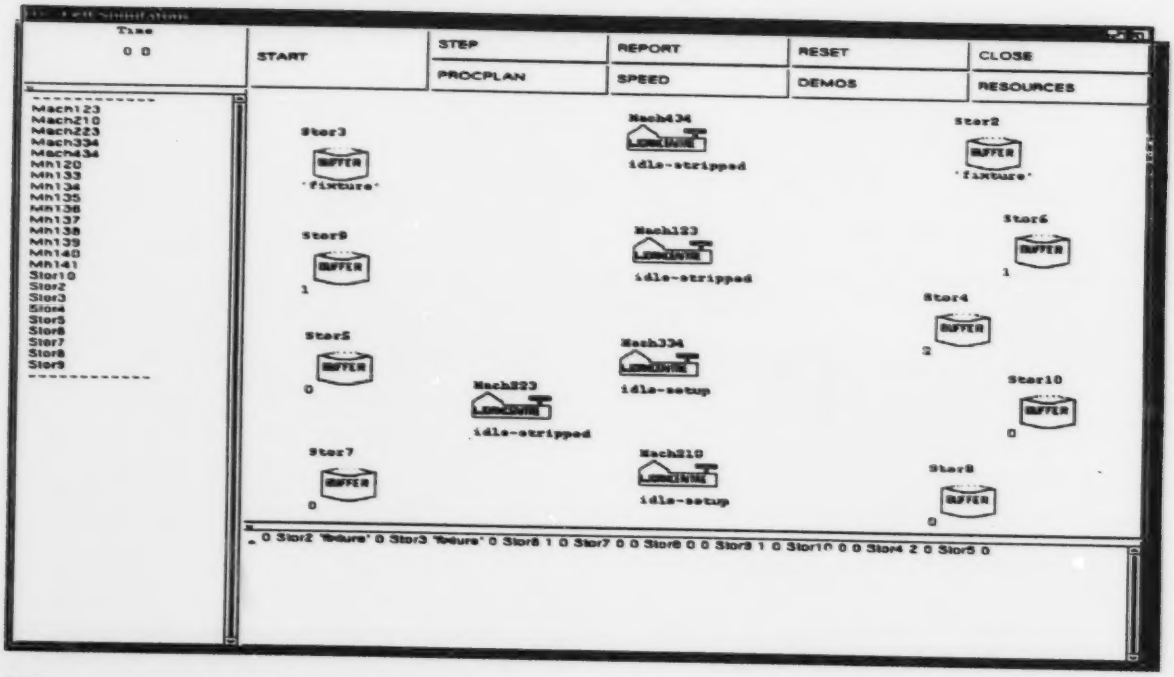


Figure 1. User interface; initial state of the example.

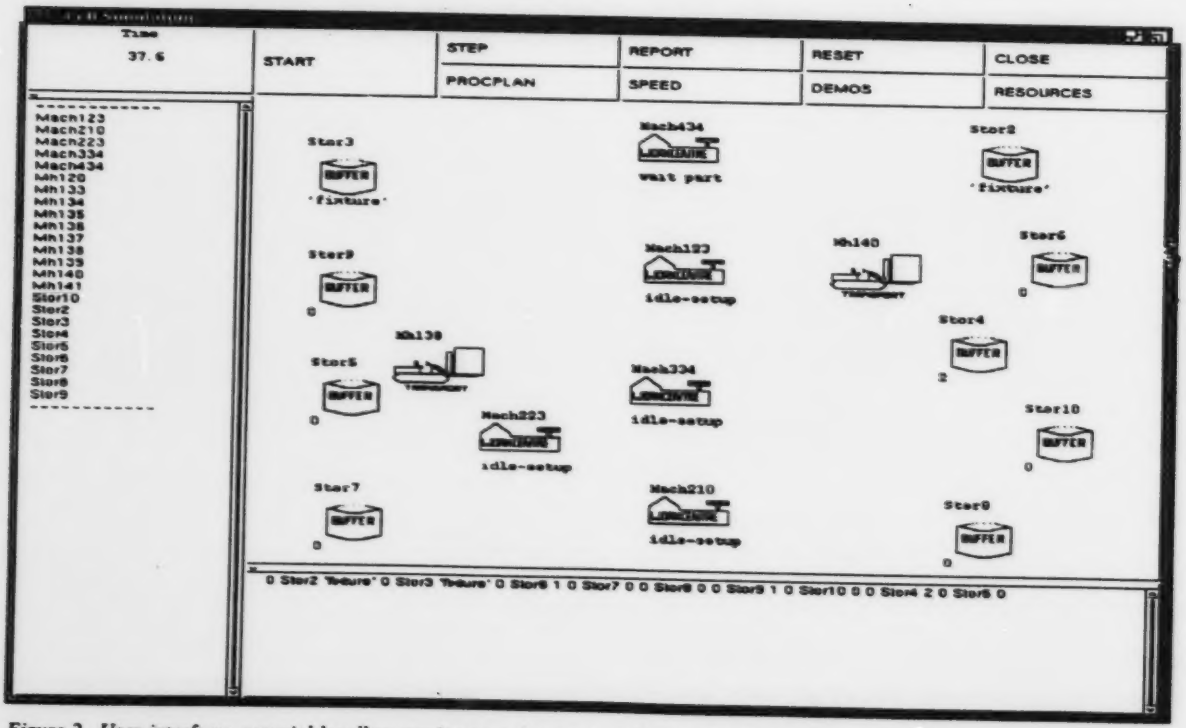


Figure 2. User interface; material handlers moving parts between machines and storage areas.

The user can choose (via the button "step") to run the model step by step or (via the button "start") to run the model from start to end of simulation, until there are no more events left in the time stack. In the latter case, the button "speed" may be used to delay the time between the occurrence of events, for a better examination of what is going on.

If, in addition to information reflected in the graphics pane, other types of information are desired to appear during the simulation, the bottom pane can be used for writing messages. The left-most pane contains a list of physical entities in the model. By clicking on an entity with the mouse button, the corresponding icon will be highlighted.

4. Comparison with the Multiprocessing Approach in Smalltalk-80

There are a number of ways discrete event simulation can be implemented in an object-oriented language and, more specifically, in Smalltalk-80. A discrete event simulation approach in Smalltalk-80 based on multiprocess scheduling is proposed by Goldberg and Robson and described in ref. 3. This approach is based on the framework used by Demos in Simula [16]. Some examples of other systems making use of this approach in Smalltalk-80 can be found in refs. 6, 10, and 11. Examples of other systems following the traditional event-based approach for discrete event simulation in an object-oriented language can be found in refs. 7–9 and 12–14. The system described in ref. 9 is written in Smalltalk-80. The others use various languages such as C++, Objective-C, Loops, and KEE. A comparison can be made between the approach based on processes and the one following the traditional discrete event simulation (such as the system here).

The approach based on processes, as described in ref. 3, uses seven main classes: SimulationObject, Simulation, DelayedEvent, Resource, which are subclasses of Object; SimulationObjec has one subclass StaticResource whereas Resource has two subclasses, ResourceProvider and ResourceCoordinator. The simulation described by these classes is also based on the event-driven approach in the sense that the simulation clock is advanced to the time at which the next action will take place.

The difference with the traditional approach is, however, that the simulation is based on processes and semaphores. A Process is a sequence of actions which can be carried out independently of the actions represented by other processes. A Semaphore allows otherwise independent processes to synchronize their actions with each other. There is a queue of delayed events containing semaphores and resume conditions (time at which the process should be resumed). When the conditions are met, the process is resumed.

The approach for discrete event simulation based on semaphores and processes can be implemented only in languages having compilers that support multiprocessing. As far as the authors know, most object-oriented languages do not use this method and simulation systems written in these languages follow the more traditional approach. The approach described in this report is a rather more traditional approach to discrete event simulation. It is more generic in the sense that the software can easily be coded in other object-oriented languages whether or not they support multiprocessing, because the concepts and ideas behind it will not change. Rather than focusing on a specific object-oriented language, the aim here has been to develop a generic system and Smalltalk-80 has been used as a typical object-oriented language to implement the concepts. If, however, the aim is to implement a discrete event simulation system in Smalltalk-80 with no future possibility and need for coding the software into another object-oriented language, it might be better to follow the multiprocessing approach with the main simulation classes having already been defined, instead of starting from scratch.

There are, however, other advantages to the present approach, such as having more flexibility and control. As a matter of fact, direct manipulation of the time stack is possible; for example, it is easy to reschedule an event or to remove it from the time stack. The amount of detail that needs to be implemented for each event, is totally flexible. This is due to the fact that each event is represented by one or more Smalltalk methods. In the multiprocessing approach, there is no correspondence between an event and a process; this type of control and manipulation of the event queue is therefore not easy. On the other hand, the approach described here demands more control written into the code. It is, however, suitable for simulation models involving unrestricted flow paths and criss-crossing of parts, such as in job shops. In the case of a well-defined flow for parts, the multiprocessing approach might be more efficient. The choice of one

approach over another depends, therefore, on the type of application.

5. Conclusions and Future Directions

It has been shown how discrete event simulation following the traditional approach using an event queue and event scheduling, can be implemented in an object-oriented environment. The language chosen for the implementation is Smalltalk-80. A discrete parts manufacturing example was used to describe this type of implementation. The approach used is, however, generic and flexible and is applicable to other discrete event simulation applications. A brief comparison was made between this approach and the implementation of discrete event simulation based on the multiprocessing approach proposed in [3]. It is planned to also apply the same framework to application domains other than manufacturing. As well, the use of Artificial Intelligence techniques in discrete event simulation is being investigated, together with its incorporation into a manufacturing enterprise model. The implementation of a rule-based framework to facilitate the evaluation of alternative decision-making rules will enhance the decision support capabilities of the Enterprise Model.

Acknowledgment

The authors would like to acknowledge the assistance of Ajit Pardasani in setting up the graphical user interface for the manufacturing example.

References

1. G.H. Beland, A.W. Chan, P.W.U. Graefe, and L.K. Nenonen. *ANEVENT: An Interactive Computer Modeling Package Based on Discrete Event Simulation*. NRC Technical Report TR-SYS-020, August 1988. NRC 29530.

2. G.S. Fishman. *Principles of Discrete Event Simulation*. John Wiley & Sons, New York, NY. 1978.

3. A. Goldberg and D. Robson. *Smalltalk-80: The Language*. Addison-Wesley. 1989.

4. B.J. Cox. *Object-Oriented Programming: An Evolutionary Approach*. Addison Wesley, Reading, MA. 1987.

5. B. Meyer. *Object-Oriented Software Construction*. Prentice Hall. 1988.

6. V.E. Knapp. The Smalltalk simulation environment Part II. *Proceedings of the 1987 Winter Simulation Conference*. Edited by A. Thesen, H. Grant, and W.D. Kelton. 1987.

7. S. Adiga and M. Gadre. Object-oriented modeling of a flexible manufacturing system. J. Intell. Robotic Syst. 3: 147–165; 1990.

8. D.L. Eldredge, J.D. McGregor, and M.K. Summers. Applying the object-oriented paradigm to discrete event simulations using the C++ language. *Simulation*, February. 1990.

9. R. Vujosevic. Object oriented visual interactive simulation. *Proceedings of the 1990 Winter Simulation Conference*. Edited by O. Balci, R.P. Sadowski, and R.E. Nance. 1990.

10. M.D. Oliff. Expert systems and intelligent manufacturing. *Proceedings of the Second International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, Charleston, SC. May 1988.

11. O.M. Ulgen. Simulation modeling in an object-oriented environment using Smalltalk-80. *Proceedings of the 1986 Winter Simulation Conference*. Edited by J. Wilson, J. Henriksen, and S. Roberts. Washington, DC. Dec. 1986.

12. M. Gittins. A system for discrete event simulation on Lisp workstations. *Third International Expert Systems Conference*, London, UK. June 1987.

13. D.P. Sanderson and L.L. Rose. Object-oriented modeling using C++. *21st Annual Simulation Symposium*, Tampa, FL. March 1988.

14. J.R. Burns and J.D. Morgeson. An object-oriented world-view for intelligent, discrete, next-event simulation. *Management Sci.* 34(12): December 1988.

15. U. Graefe and H. Atabakhsh. Use of modeling in the specification of a manufacturing language environment. *Proceedings of The Symposium on Manufacturing Application Programming Language Environments*, Ottawa, Ont. May 14–15, 1990. pp. 147–153.

16. G.M. Birtwistle. *A System for Discrete Event Modeling in Simula*. MacMillan 1979.

Appendix A: List of Entities and their Attributes for the Model

- Machine: represents the resources used in the model. It has attributes such as:
  - machine identity
  - status:
    - idle-stripped down
    - setup in progress
    - waiting for part
    - machining
    - part waiting for removal
    - idle-setup
    - teardown in progress
    - non-operational (down)
  - part on/off machine
  - time status was assumed
  - current job identity
  - current operation identity
  - current schedule entry identity
- Storage Area: stores elements such as fixtures, raw parts, and final products. It has attributes such as:
  - identity
  - status: full or not full
  - current total count: number of elements it contains
  - statistics information (for each job identity):
    - operation identity
    - job identity
    - current part count
    - status for this job/operation
    - time of last update
- Material Handler: moves fixtures, raw parts, and final products between the devices (storage areas and machines). The attributes are:
  - material handler identity
  - status: idle or busy
  - current job identity
  - last operation identity
  - from device identity
  - to device identity
  - t0: time for proceeding to source device location (nominal value)
  - t1: time for picking up part (n.v)
  - t2: time for depositing part (n.v.)
  - t3: time for returning to standby position (n.v.)
  - nominal speed
  - task queue
  - current task type
  - start time for current task
  - end time for current task
  - statistics: number of setup operations, teardown operations, machining operations done, etc.
- Operation: corresponds to a machining operation. Its attributes are:
  - operation identity
  - preceding operation
  - succeeding operation
  - device identity
  - duration (machining time)
  - material handler identity (to transfer part to the next operation)
  - job identity
  - current quantity completed
  - status:
    - not yet started
    - waiting for part
    - proceeding
    - part waiting for removal
    - all parts completed
- Setup/Teardown: correspond to a setup/teardown operation. They have the following attributes.
  - setup/teardown identity
  - machine identity
  - duration of operation
  - material handler identity
  - fixture storage area identity
- MachineSchedule: represents the schedule for a machine. It has the following attributes:
  - machine identity
  - current schedule entry
- Schedule: represents one entry in the schedule (one column) and has the following attributes:
  - scheduled start time
  - scheduled end time
  - simulated start time
  - simulated end time
  - job identity
  - operation type
  - operation/setup/teardown identity
  - machine identity
- Job: corresponds to a job in the model and has the following attributes:
  - job identity
  - total number of operations in the job
  - total number of parts to manufacture

- Task: a material handling task is created when it is issued to the material handler. It has the following attributes:
  - source device identity
  - sink device identity
  - operation identity
  - time it was created
  - start time
  - end time
  - material handler identity
- Bill of Operations: corresponds to a bill of operations for a job and has the job identity as its attributes.
- Time Stack: contains the events that are scheduled to happen and the corresponding event times (in chronological order).
- Machine List: list of machines to be scheduled.

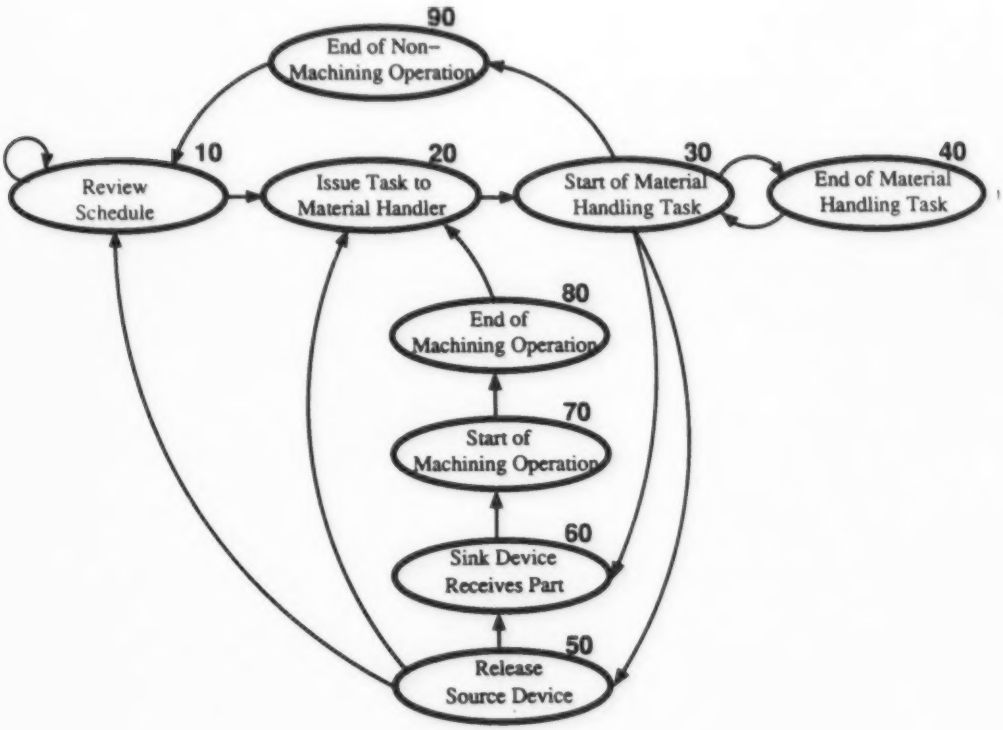
Appendix B: Hierarchy of Smalltalk Objects for the Enterprise Model

```
Object ()
 Model ("dependents")
   EnterpriseObject ()
     Gam ()
        Equipment ("location" "iconSize")
          Device ()
     Machine ("machId" "status" "currentJob" "currentOp" "curMachOp" "currSchedEnt" "statusTime"
"partOnMach" "schedId")
              StorageArea ("storagId" "status" "totalCount" "stat")
 MaterialHandler ("matHandId" "status" "curSched" "lastOp" "fromDevice" "toDevice" "taskQ" "t0" "t1" "t2"
"t3" "speed" "curTaskTy" "curTaskDur" "totSet" "totTear" "totMach" "totLongTask" "totShortTask" "dis-
playFlag")
     Interface ()
     Job ("jobId" "firstOp" "totalOp" "totalParts" "booId")
 Operation ("operId" "precedOp" "succOp" "deviceId" "duration" "matHandlerId" "quanComplete" "jobId" "sta-
tus")
 Schedule ("startTime" "endTime" "opType" "jobId" "opId" "machId" "setupId" "tearId" "simStart" "simEnd")
     Setup ("setupId" "machId" "duration" "matHandlerId" "storagId")
     StorageInfo ("opId" "jobId" "partCnt" "timeUpdate" "statusOp")
     Task ("source" "sink" "operType" "operId" "matHandler" "creatime" "startime" "endtime")
          Teardown ("tearId" "machId" "duration" "matHandlerId" "storagId")

Object ()
        Model ("dependents")
            Cell ("components" "machList" "list" "currentObject" "addFlag" "remove Flag"
"listSelection" "timeStack" "text" "delayTime" "animateFrom" "animateTo")

Object ()
        Collection ()
            SequenceableCollection ()
                OrderedCollection ("firstIndex" "lastIndex")
                    BillOperation ("jobId")

Object ()
        Collection ()
            SequenceableCollection ()
                OrderedCollection ("firstIndex" "lastIndex")
                    MachineList ()

Object ()
        Collection ()
            SequenceableCollection ()
                OrderedCollection ("firstIndex" "lastIndex")
                    MachSchedule ("machId" "curIndex" "curSchedEntry")

Object ()
        Collection ()
            SequenceableCollection ()
```

```
                    OrderedCollection ("firstIndex" "lastIndex")
                        StorageStat ()
Object ()
        Collection ()
            SequenceableCollection ()
                OrderedCollection ("firstIndex" "lastIndex")
                    Table ()

Object ()
        Collection ()
            SequenceableCollection ()
                OrderedCollection ("firstIndex" "lastIndex")
                    SortedCollection ("sortBlock")
                        TimeStack ("indexList")
```

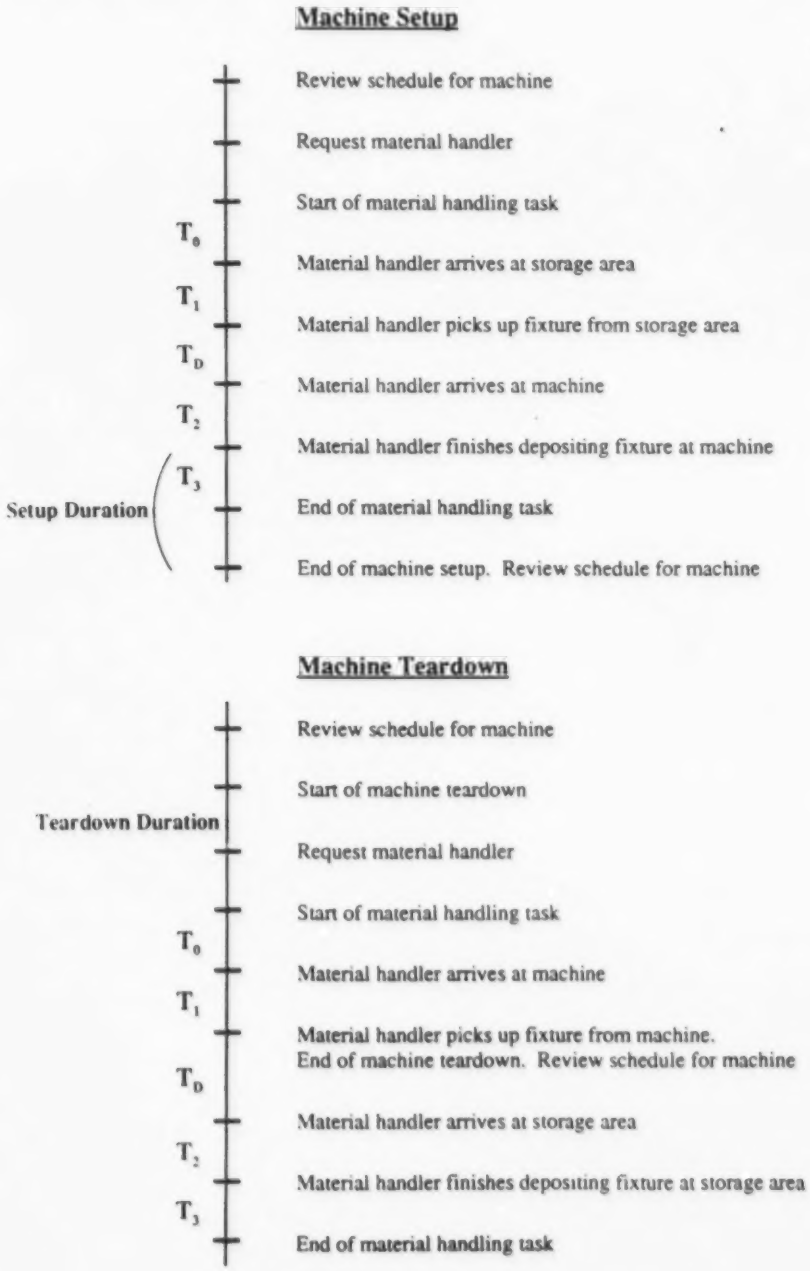
Appendix C: Flowcharts for the Events

C1. Event Hierarchy

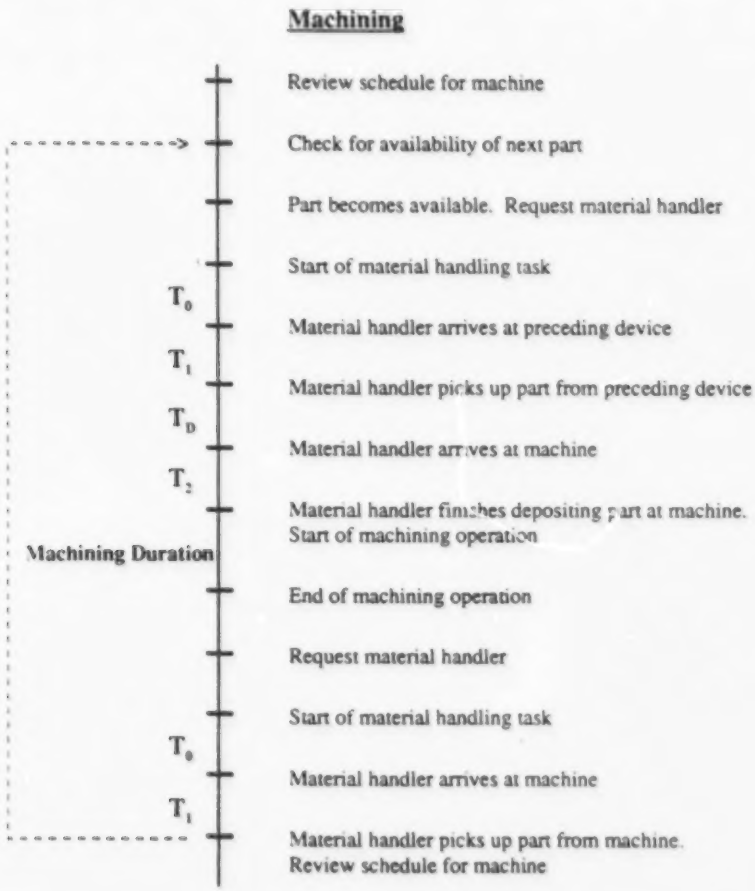


Appendix D: Description of the Flowcharts

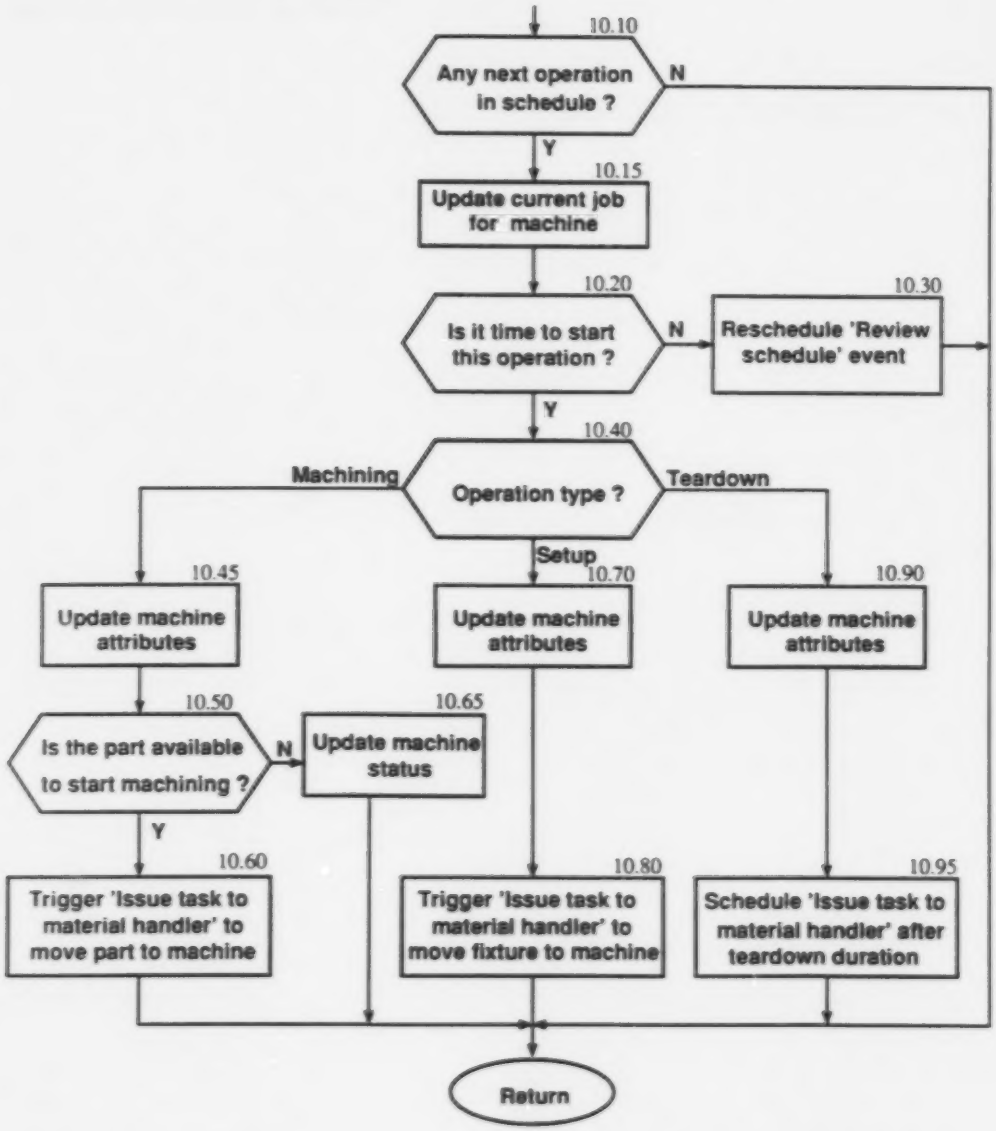
D1. Event Sequence — Part 1



D2. Event Sequence — Part 2



D3. Event — Review Schedule



10.10 ANY NEXT OPERATION?

The schedule for this machine is scanned to see if there is any more operation left.

10.15 UPDATE CURRENT JOB FOR MACHINE

The current-job attribute of the machine is updated according to the job ID provided by the schedule.

10.20 IS IT TIME TO START THIS OPERATION?

Pass the job number to the machine attributes. Check the scheduled time for the next operation of this job.

10.30 RESCHEDULE "REVIEW SCHEDULE" EVENT

If it is not yet time for the next operation, reschedule this event to occur at the scheduled time for the next operation.

10.40 OPERATION TYPE?

Check the operation type for the next operation.

10.45 UPDATE MACHINE ATTRIBUTES

The current-operation attribute of the machine is updated according to the operation ID provided by the schedule.

10.50 IS PART AVAILABLE?

If the next operation is a machining operation, check if the part is available to start machining. This is done by checking the previous operation for this job in the bill of operations. If the previous device is a machine, the status of the machine will indicate whether the part is ready for this machine. If the device is a storage area, the storage summary information will indicate if the part is available.

10.60 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

Set up a material handling task with the current machine being the sink and the operation-type being "machining". Trigger "Issue task to material handler" event to give task to material handler to move the part from the previous device to this machine.

10.65 UPDATE MACHINE STATUS

The machine status is set to "waiting for part."

10.70 UPDATE MACHINE ATTRIBUTES

The machine status is set to "setup in progress." The current-operation for the machine is set to the setup ID.

10.80 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

Set up a material handling task with the current machine being the sink and the operation-type being "setup." Trigger "Issue task to material handler" event to give task to material handler to move fixture from its storage area to this machine.

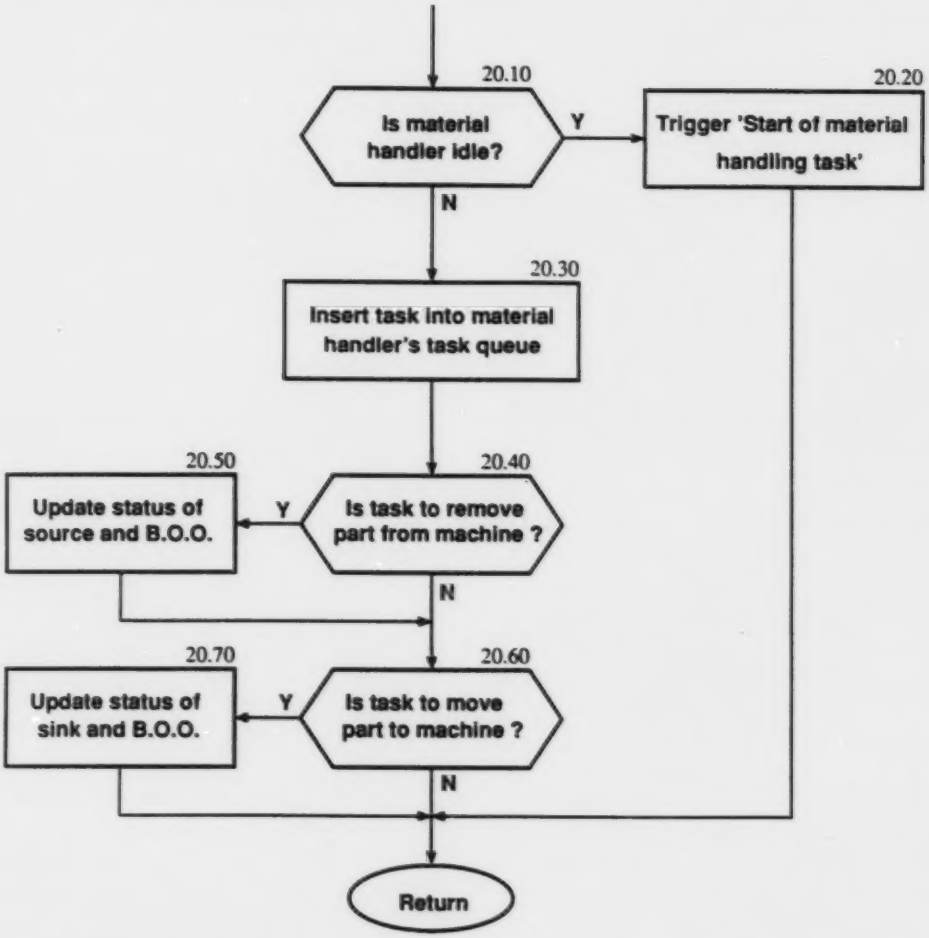
10.90 UPDATE MACHINE STATUS

The machine status is set to "teardown in progress." The current-operation for the machine is set to the teardown ID.

10.95 SCHEDULE "ISSUE TASK TO MATERIAL HANDLER" EVENT

Set up a material handling task with the current machine being the source and the operation-type being "teardown." Schedule "Issue task to material handler" event for material handler to move fixture from this machine to its storage area after the teardown duration.

D4. Event: Issue Task to Material Handler



20.10 IS MATERIAL HANDLER IDLE?

The status of the material handler is checked to see if it is idle.

20.20 TRIGGER "START OF MATERIAL HANDLING TASK" EVENT

If the material handler is idle, then the material handling task is assigned to the material handler and it can start right away.

20.30 INSERT TASK INTO MATERIAL HANDLER'S TASK QUEUE

If the material handler is busy, the material handling task is inserted into the material handler's task queue to await execution.

20.40 IS TASK TO REMOVE PART FROM MACHINE?

The task just issued is checked to see if it involves removing a part from a machine, i.e., the operation-type

must be "machining" and the source must be a machine.

20.50 UPDATE STATUS OF SOURCE AND BILL OF OPERATIONS

If the task involves removing a part from a machine, the machine status for the source is set to "part waiting to be removed." The operation status is set to "part completed but waiting."

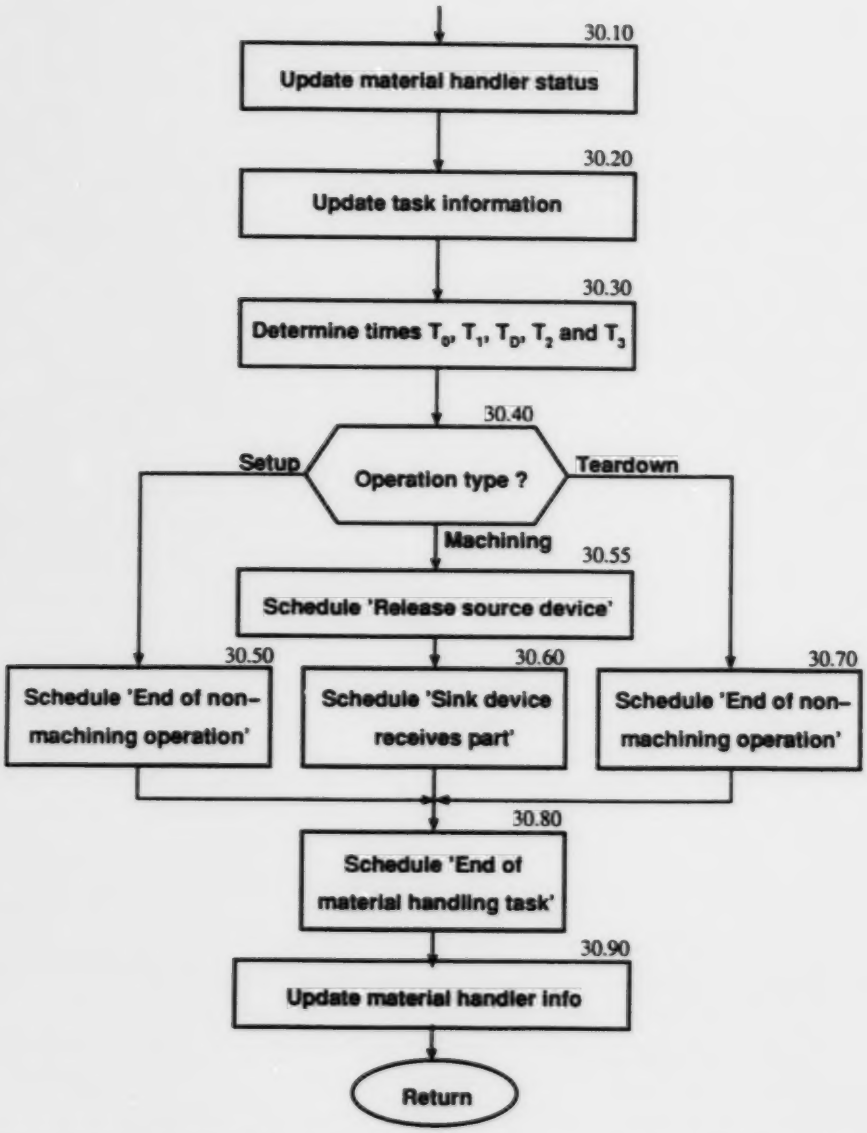
20.60 IS TASK TO MOVE PART TO MACHINE?

The task is also checked to see if it involves moving a part to a machine, i.e., the operation-type must be "machining" and the sink must be a machine.

20.70 UPDATE STATUS OF SINK AND BILL OF OPERATIONS

If the task involves moving a part to a machine, the machine status for the sink is set to "waiting for part." The operation status is also set to "waiting for next part."

D5. Event: Start of Material Handling Task



30.10 UPDATE MATERIAL HANDLER STATUS

The status of the material handler is set to "busy."

30.20 UPDATE TASK INFORMATION

The start-time for the task is updated to the current time.

30.30 DETERMINE TIMES T0, T1, TD, T2, AND T3

Determine time T0 for the material handler to get to the source device, time T1 to pick up the part (or fixture), time TD to travel to the sink device (based on the distance between the source and sink devices), time T2 to deposit the part, and time T3 for the material handler to return to standby.

30.40 OPERATION TYPE?

Check the operation type for the applicable operation.

30.50 SCHEDULE "END OF NON-MACHINING OPERATION" EVENT

If the applicable operation is a setup operation, schedule the "End of non-machining operation" event to occur (T0 + T1 + TD + T2 + setup duration) from now.

30.55 SCHEDULE "RELEASE SOURCE DEVICE" EVENT

The "Release source device" event is scheduled to occur at time (T0 + T1) from now if the source device is a machine.

30.60 SCHEDULE "SINK DEVICE RECEIVES PART" EVENT

If the applicable operation is a machining operation, schedule the "Sink device receives part" event to occur (T0 + T1 + TD + T2) from now.

30.70 SCHEDULE "END OF NON-MACHINING OPERATION" EVENT

If the applicable operation is a teardown operation, schedule the "End of non-machining operation" event to occur (T0 + T1 + TD + T2) from now.

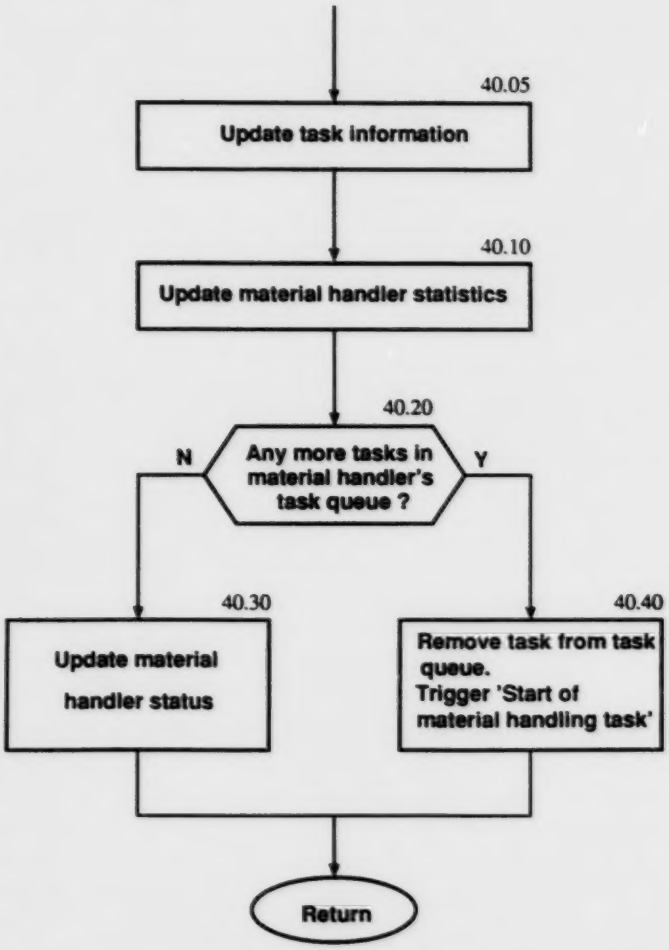
30.80 SCHEDULE "END OF MATERIAL HANDLING TASK" EVENT

The "End of material handling task" event is scheduled to occur (T0 + T1 + TD + T2 + T3) from now.

30.90 UPDATE MATERIAL HANDLER INFORMATION

The current-operation, source-device and sink-device for the material handler are all updated.

D6. Event: End of Material Handling Task



40.05 UPDATE TASK INFORMATION

The task end-time is updated to the current time.

40.10 UPDATE MATERIAL HANDLER STATISTICS

The statistics (on task type, task duration, etc.) for the material handler is updated.

40.20 ANY MORE TASKS?

The task queue for the material handler is checked to see if there are any more tasks to be executed by this material handler.

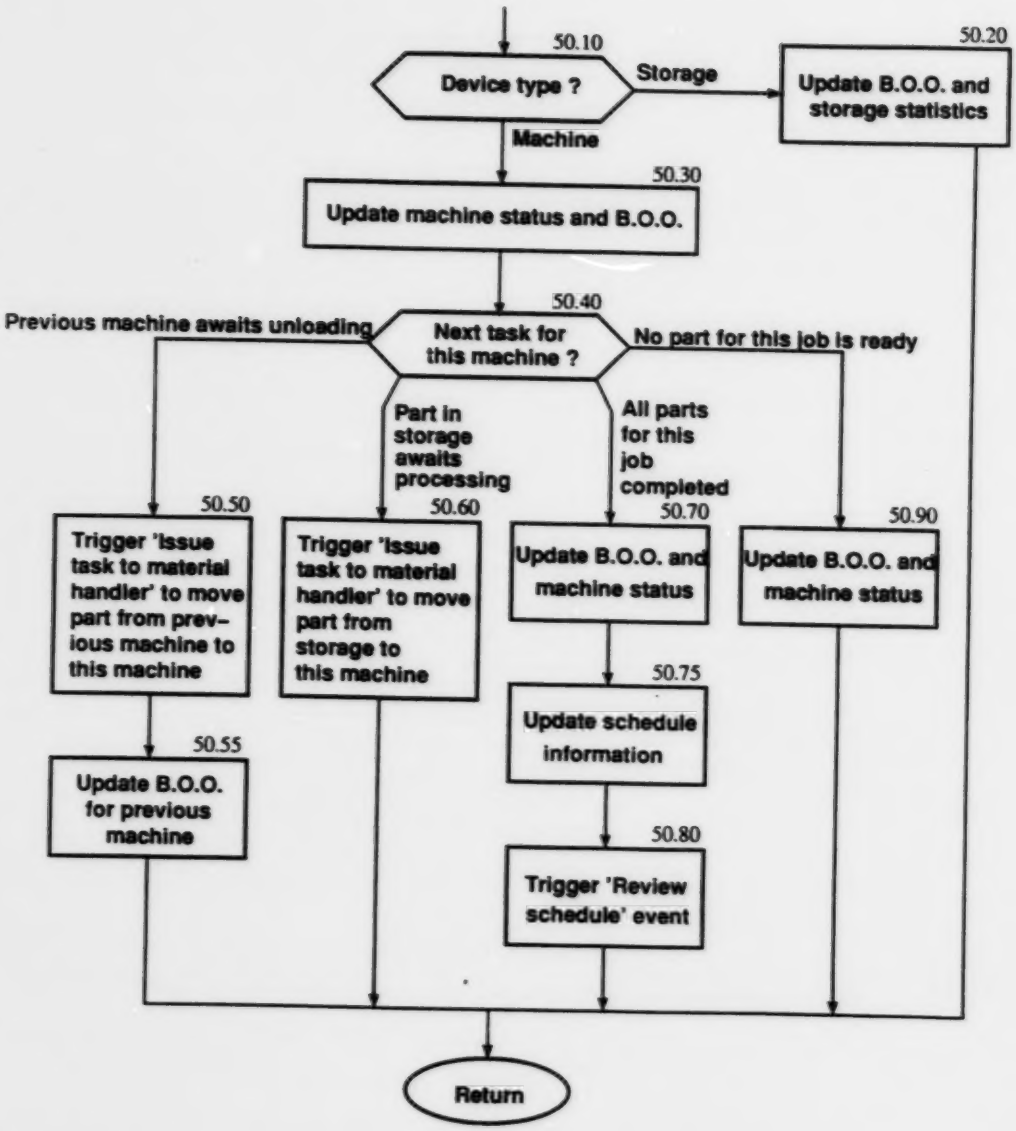
40.30 UPDATE MATERIAL HANDLER STATUS

If there are no more tasks in the task queue, the status of this material handler is set to "idle."

40.40 REMOVE TASK FROM TASK QUEUE

If there are more tasks in the task queue, the next task is selected and removed from the queue for execution. Trigger the "Start of material handling task" event to occur now.

D7. Event: Release Source Device



50.10 DEVICE TYPE?

The device type of the source device is checked. It can be a machine or a storage area.

50.20 UPDATE BILL OF OPERATIONS AND STORAGE STATISTICS

If the source device is a storage area, the bill of operations and the summary information for the storage

area are updated to reflect the part which has just been removed.

50.30 UPDATE MACHINE STATUS AND BILL OF OPERATIONS

If the source device is a machine, the machine status is set to "idle—setup." The number of parts and the status of the appropriate operation in the bill of operations are updated to reflect the part which has just been removed. If the status of the operation is "waiting for part removal," it is set to "proceeding."

50.40 NEXT ACTIVITY FOR THIS MACHINE?

The bill of operations for the current job is checked to see if there are any parts from the previous operation waiting. There are four possible results to this check.

50.50 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

A task is issued to the material handler to move the part from the previous machine to this machine.

50.55 UPDATE BILL OF OPERATIONS FOR PREVIOUS OPERATION

The status of the previous operation in the bill of operations is set from "part completed but waiting" to "proceeding."

50.60 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

A task is issued to the material handler to move the part from the storage area to this machine.

50.70 UPDATE BILL OF OPERATIONS AND MACHINE STATUS

If all the parts for this job have been completed, then the status of the current operation is set to "all parts completed," and the status of the machine remains "idle—setup."

50.75 UPDATE SCHEDULE INFORMATION

The simulated end time for the current schedule entry is set to the current time.

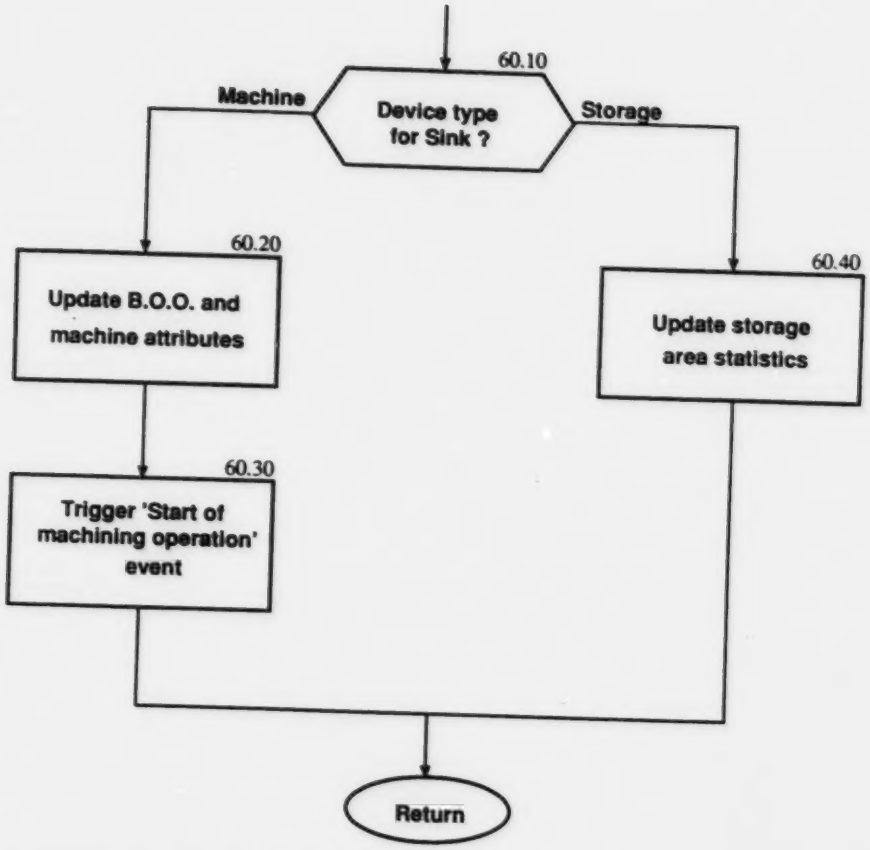
50.80 TRIGGER "REVIEW SCHEDULE" EVENT

Since the current operation in the machine schedule is now completed, the "Review Schedule" event is triggered to proceed to the next operation in the schedule.

50.90 UPDATE BILL OF OPERATIONS AND MACHINE STATUS

If no part for this job is ready, then the statuses of the current operation and machine are both set to "waiting for part."

D8. Event: Sink Device Receives Part



60.10 DEVICE TYPE FOR SINK?

The device type of the sink device is checked. It can be a machine or a storage area.

60.20 UPDATE BILL OF OPERATIONS AND MACHINE ATTRIBUTES

If the sink device is a machine, the operation status in the bill of operations is set to "proceeding." The operation number for the machine is updated if necessary, and the machine attribute is updated to reflect the part which has just been delivered.

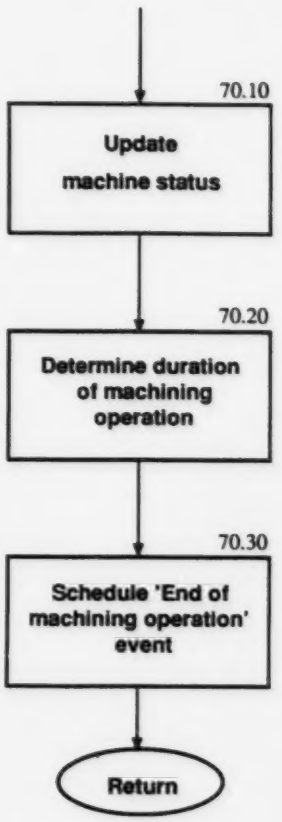
60.30 TRIGGER "START OF MACHINING OPERATION" EVENT

Trigger the "Start of machining operation" event to occur now for this machine.

60.40 UPDATE STORAGE AREA STATISTICS

If the sink device is a storage area, the count for storage is incremented. The summary information is also updated to reflect the newly added part type and current operation.

D9. Event: Start of Machining Operation



70.10 UPDATE MACHINE STATUS

The status of the machine is set to "machining."

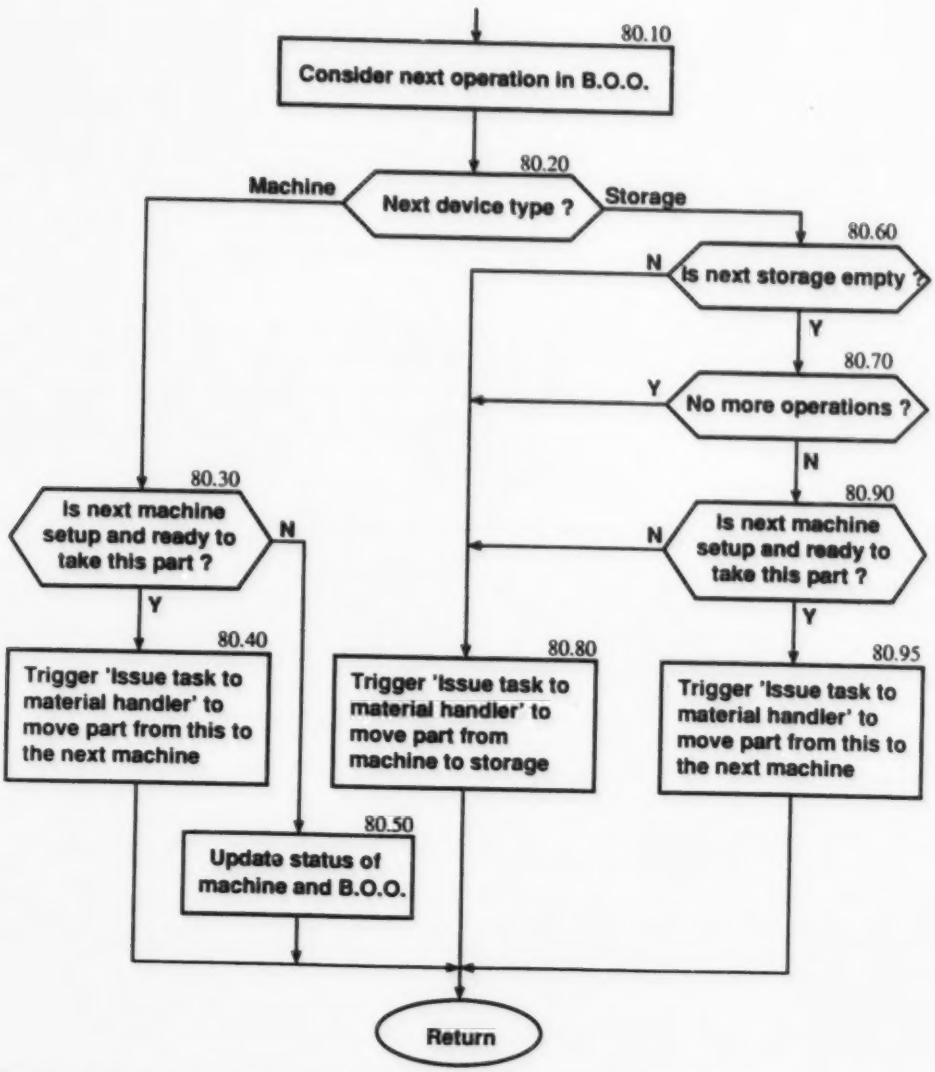
70.20 DETERMINE DURATION OF MACHINING OPERATION

The duration for the machining operation is obtained from the bill of operations or determined from a statistical distribution.

70.30 SCHEDULE "END OF MACHINING OPERATION" EVENT

The "End of machining operation" event is scheduled to occur after the duration of the machining operation.

D10. Event: End of Machining Operation



80.10 CONSIDER NEXT OPERATION IN BILL OF OPERATIONS

From the bill of operations, determine the next operation for this part. (The assumption is made here that the last machining operation must be followed by a storage area.)

80.20 NEXT DEVICE TYPE?

Check what type of device is the next device. It can be a machine or a storage area.

80.30 IS NEXT MACHINE SETUP AND READY TO TAKE THIS PART?

If the next device is a machine, its status is checked to see if it is setup for this part and if it is ready to take it. The status must be either "idle—setup," or "waiting for part." The job ID for the next machine is also verified that it corresponds to the current job.

80.40 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

If the next machine is ready to take the part, a material handling task is issued to the material handler to move the part from this machine to the next machine.

80.50 UPDATE STATUS OF MACHINE AND BILL OF OPERATIONS

If the next machine is not ready to take this part, the status of this machine is set to "part waiting to be removed," and the operation status in the bill of operations is set to "part completed but waiting."

80.60 IS NEXT STORAGE EMPTY?

If the next device is a storage area, the summary information for this storage area is checked to see if it is empty with respect to this part type.

80.70 NO MORE OPERATIONS?

If the next storage area is empty, the bill of operations is checked to see if there is a next operation beyond the storage. If there is, it has to be a machining operation, and the status of that machine is checked to see if it can take this part right away. If there is no more operation beyond the next storage area, then the part is sent to the storage area.

80.80 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

If the next storage is not empty (i.e., there are already parts of this type waiting for the next operation), a task is issued to the material handler to move the part from the machine to the storage area.

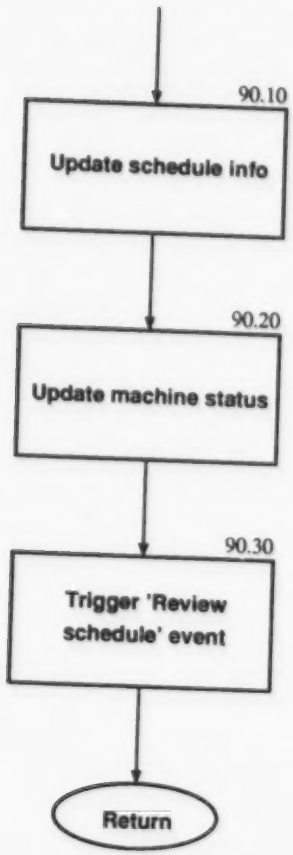
80.90 IS NEXT MACHINE SETUP AND READY TO TAKE THIS PART?

If there are operations subsequent to the storage, the next machine is checked to see if it is setup for this part and if it is ready to take it.

80.95 TRIGGER "ISSUE TASK TO MATERIAL HANDLER" EVENT

If the machine beyond the next storage is ready to take the part, a material handling task is issued to the material handler to move the part from this machine to that machine directly. Otherwise, the part goes into the next storage.

D11. Event: End of Non-machining Operation



90.10 UPDATE SCHEDULE INFORMATION

The simulated end time of a setup or teardown operation is noted in the schedule information.

90.20 UPDATE MACHINE STATUS

If the non-machining operation just completed is a setup, the status of the machine is set to idle—setup. If the operation is a teardown, the status of the machine is set to idle—stripped down.

90.30 TRIGGER "REVIEW SCHEDULE" EVENT

Trigger the "Review schedule" event to proceed to the next operation for this machine in the schedule.

D12. Documentation and Smalltalk Programs Cross-reference List

Event No.	Subclass	Method	Block No.
10	#MachSchedule	doReviewSched	10.10
	#Schedule	reviewSched	10.15 – 10.40
		machinOper	10.45 – 10.65
		setupOper	10.70 – 10.80
		tearDwOper	10.90 – 10.95
20	#MaterialHandler	issueTask	20.10 – 20.70
30	#MaterialHandler	startMatHandling	30.10 – 30.30
			30.80 – 30.90
	#Setup	startMat	30.50
	#Operation	startMat	30.55 – 30.60
	#Teardown	startMat	30.70
40	#MaterialHandler	endMatHandling	40.05 – 40.40
50	#MachSchedule	releaseSourceDev	50.10
	#StorageArea	release	50.20
	#Machine	release	50.30 – 50.90
60	#MachSchedule	sinkDevReceive	60.10
	#Machine	receive	60.20 – 60.30
	#StorageArea	receive	60.40
70	#MachSchedule	startMachOper	70.10 – 70.30
80	#MachSchedule	endMachOper	80.10 – 80.20
	#Machine	endMach	80.30 – 80.50
	#StorageArea	endMach	80.60 – 80.95
90	#MachSchedule	endNonMachOp	90.10 – 90.30

Appendix E. Input Data for Initializing the Enterprise Model Example

This example corresponds to demo1 and consists of three jobs, three bill of operations, five machines, and three schedules each for a given machine. The schedules share resources and run in parallel.

The following is a list of objects initialized for the example. These objects together with all other objects in the model can be found in the file hierarchy of objects (4.2).

E1. Schedules for Machines

Schedule for Machine Mach123:

Entry no.:	1	2	3
Sched. start time:	8	9	10
Sched. end time:	9	10	11
Operation type:	su	mach	td
Job id.:	j1089	j1089	j1089
Operation identity:	—	op6	—
Setup/Teardown id.:	su4	—	td5

Schedule for Machine Mach434:

Entry no.:	1	2	3
Sched. start time:	12	13	14
Sched. end time:	13	14	15
Operation type:	su	mach	td
Job id.:	j2089	j2089	j2089
Operation identity:	—	op31	—
Setup/Teardown id.:	su6	—	td7

Schedule for Machine Mach223:

Entry no.:	1	2	3
Sched. start time:	21	23	25
Sched. end time:	23	25	26
Operation type:	su	mach	td
Job id.:	j1035	j1035	j1035
Operation identity:	—	op20	—
Setup/Teardown id.:	su8	—	td9

E2. Bill of Operations for Jobs

Bill of Operations for Job j1089:

Oper id.:	op5	op6	op7	op8	op9	op10
Predecessor:	—	op5	op6	op7	op8	op9
Successor:	op6	op7	op8	op9	op10	—
Device id.:	stor6	mach123	stor7	mach334	mach210	stor8
Mach. time:	—	3	—	1	2	—
Mat hand. id.:	mh140	mh141	mh140	mh137	mh136	—

Bill of Operations for job j1035:

Oper id.:	op19	op20	op21
Predecessor:	—	op19	op20
Successor:	op20	op21	—
Device id.:	stor9	mach223	stor10
Mach. time:	—	2	—
Mat hand. id.:	mh138	mh139	—

Bill of Operations for job j2089:

Oper id.:	op30	op31	op32
Predecessor:	—	op30	op31
Successor:	op31	op32	—
Device id.:	stor4	mach434	stor5
Mach. time:	—	3	—
Mat hand. id.:	mh140	mh141	—

E3. Other Objects

Machine:

mach. id.:	mach123	mach223	mache434
status:	idle-stripped	idle-stripped	idle-stripped
mach. id.	mach334	mach210	
status	idle-setup	idle-setup	

Setup/Teardown:

Setup/teardown id.:	su4	td5	su8	td9	su6	td7
Mach id.:	mach123	mach123	mach223	mach223	mach434	mach434
Mat hand id.:	mh120	mh120	mh133	mh133	mh120	mh120
Duration:	9	8	3	2	3	5
Storage Area id.:	stor3	stor3	stor2	stor2	stor3	stor3

Job:

job id.:	j1089	j1035	j2089
first oper:	op5	op19	op30
total oper.:	3	3	3
total parts:	1	1	2

Storage Area:

storage id.:	stor2	stor3	stor4	stor5	
status:	not full	not full	not full	not full	
total count:	fixtures	fixtures	2	0	
tatus: id.:	stor6	stor7	stor8	stor9	stor10
status:	not full	not full	not full	not full	not full
total count:	1	0	0	1	0

Material Handler:

mat. hand. id.:	mh120	mh133	mh136	mh137
status:	idle	idle	idle	idle
speed:	1	5	2	3
t0:	4	3	1	2
t1:	3	2	3	1
t3:	2	5	3	3
mat. hand. id.:	mh138	mh139	mh140	mh141
status:	idle	idle	idle	idle
speed:	1	1	3	2
t0:	2	4	4	2
t1:	3	2	1	4
t2:	1	1	3	3
t3:	2	2	3	1

Distance Table:

Contains entries corresponding to machines and storage areas, and the distances between them. The value of travel time (td) is calculated fram the relevant value found in this table and the speed of the corresponding material handler.